



Persistent

Core Java : Streams And Files

Persistent Interactive | Persistent University



Objectives:

- At the end of this module, you will be able to understand & implement :
 - Basic concepts of IO streams
 - Use of File and File Handling
 - Serialization and Deserialization
 - Serialization and Composition
 - Serialization and Inheritance
- ARM (Automatic Resource Management)
- Console Class

Overview of IO streams

- An I/O Stream represents an input source or an output destination.
- A stream can represent many different kinds of sources and destinations, like
 - Disk Files, Devices, Other Programs, and Memory Arrays.
- Streams support different kinds of data including
 - Bytes, Primitive Data Types, Localized Characters, and Objects.
- Some streams simply pass on data or manipulate and transform the data in useful ways.

Classification of IO streams

- **Byte Streams** handle I/O of raw binary data.
- **Character Streams** handle I/O of character data, automatically handling translation to and from the local character set.
- **Buffered Streams** optimize input and output by reducing the number of calls to the native API.
- **Scanning and Formatting** allows a program to read and write formatted text.
- **Command Line IO** describes the Standard Streams and the Console object.
- **Data Streams** handle binary I/O of primitive data type and String values.
- **Object Streams** handle binary I/O of objects.

More about IO

- Input / Output Stream and sub-classes were part of Java 1.1.
- Byte Streams operates on 8 bit (1 byte) data.
- Classes for Byte IO are called as Input Streams and Output Streams
- Java 1.2 adds more classes specialized for character based I/O
 - The stream classes are for data I/O.
- Classes for character I/O are called Readers and Writers
- Character Streams Operates on 16-bit (2 byte) Unicode characters.
- These classes deal with streams of characters
 - Read/write single character or array of characters
 - Again there are classes specialised for particular purposes

Unicode

- Each character in the ASCII character set fits into a single byte
 - but that's not enough for Chinese, and other complex alphabets
 - Need more than a single byte
 - A Java character (char) is 2 bytes
- Java handles text using Unicode
 - International standard character set, containing characters for almost all known languages
 - And a few imaginary ones! (Klingon, Elvish...)
- Inside the JVM all text is held as Unicode

Overview of File and File Handling

- **I/O classes you'll need to understand**
 - File
 - FileReader
 - BufferedReader
 - FileWriter
 - BufferedWriter
 - PrintWriter

The File class

- General purpose utility class for working with files and directories.

`/* represents the path of a file */`

`File file1 = new File("C:\Windows\win.ini");`

`/* represents the path of either a file or directory both of which may or may not exist */`

`File file2 = new File("C:\DirA\DirB");`

`/* represents a path as a combination of parent and child elements */`

`File file3 = new File("C:\\Program Files", "xerox");`

Creating files using Class file

//There is no file as yet

```
import java.io.*;
```

```
class Writer1 {
```

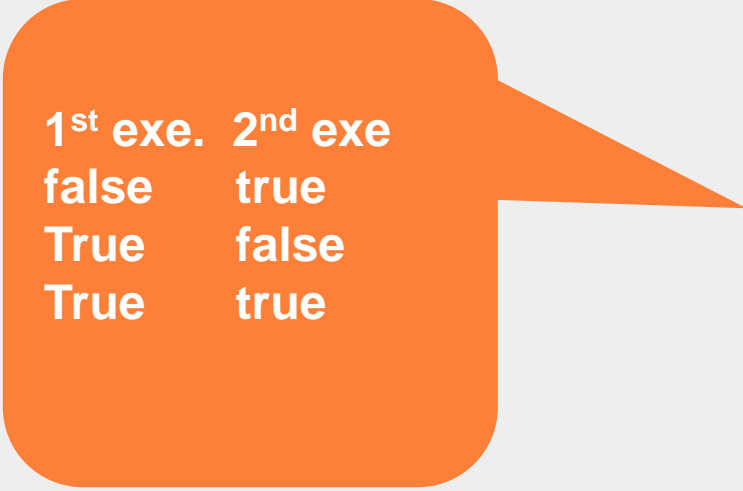
```
    public static void main(String [] args) {
```

```
        File file = new File("fileWrite1.txt");
```

```
    }
```

```
}
```

Creating files using Class file



1 st exe.	2 nd exe
false	true
True	false
True	true

```
import java.io.*;

class Writer1 {

    public static void main(String [] args) {

        try {
            boolean newFile = false;

            File file = new File("fileWrite1.txt");

            System.out.println(file.exists());

            newFile = file.createNewFile();

            System.out.println(newFile);

            System.out.println(file.exists());

        } catch(IOException e) { }

    }
}
```

Using FileWriter and FileReader

**//creates an actual file
& a FileWriter obj**

```
import java.io.*;

class Writer2 {

    public static void main(String [] args) {

        char[] in = new char[50];

        int size = 0;

        try {

            File file = new File("fileWrite2.txt");

            FileWriter fw =new FileWriter(file);

            fw.write("howdy\nfolks\n");
```

Using FileWriter and FileReader (Contd.)

```
fw.flush();

fw.close();

FileReader fr = new FileReader(file);

size = fr.read(in);

System.out.print(size + " ");

for(char c : in) // print the array
    System.out.print(c);
fr.close();

}
catch(IOException e) { }

}

}
```

Classes Summary

java.io Class	Extends From	Key Constructor(s) Arguments	Key Methods
File	Object	File, String String String, String	createNewFile() delete() exists() isDirectory() isFile() list() mkdir() renameTo()
FileWriter	Writer	File String	close() flush() write()
BufferedWriter	Writer	Writer	close() flush() newLine() write()
PrintWriter	Writer	File (as of Java 5) String (as of Java 5) OutputStream Writer	close() flush() format()*, printf()*, print(), println() write()
FileReader	Reader	File String	read()
BufferedReader	Reader	Reader	read() readLine()
			*Discussed later

Combining I/O Classes

```
File file = new File("fileWrite2.txt"); // create a File
```

```
PrintWriter pw = new PrintWriter(file); // pass file to the  
//PrintWriter
```

```
File file = new File("fileWrite2.txt"); // create a File
```

```
// create a FileWriter that will send its o/p to file
```

```
FileWriter fw = new FileWriter(file);
```

```
// create a PrintWriter that will send its o/p to Writer
```

```
PrintWriter pw = new PrintWriter(fw);
```

```
pw.println("howdy");
```

```
pw.println("folks");
```

Combining I/O Classes

To Read Data

```
// create a File and open fileWrite2.txt
```

```
File file = new File("fileWrite2.txt");
```

```
// create a FileReader to get data from file
```

```
FileReader fr = new FileReader(file);
```

```
// create a BufferedReader to get its data from a Reader
```

```
BufferedReader br = new BufferedReader(fr);
```

```
//BufferedReader
```

```
String data=br.readLine();
```

Working with files and Directories

```
File file = new File("foo");
```

- Two ways to create a file:
 1. Invoke the `createNewFile()` method on a `File` object. For example:

```
file.createNewFile();
```

2. Create a `Reader` or a `Writer` or a `Stream`.

```
PrintWriter pw = new PrintWriter(file);
```


Creating a Directory

- Way to create a directory:
- Invoke the mkdir() method on a File object. For example.
- With readers and writers it is not possible to create directory.
- Exception occurs if directory does not exist & and we try to create file.

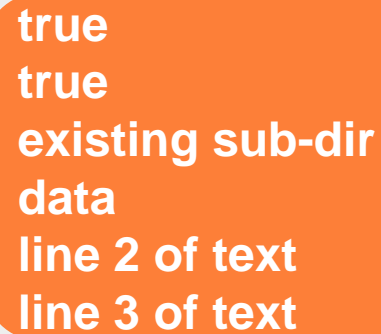
```
File myDir = new File("mydir");
```

```
myDir.mkdir();
```

```
File file = new File(myDir, "myFile.txt");
```

```
file.createNewFile();
```

Creating a Directory



true
true
existing sub-dir
data
line 2 of text
line 3 of text

```
File existingDir = new File("existingDir");
```

```
System.out.println(existingDir.isDirectory());
```

```
File existingDirFile = new File(  
existingDir, "existingDirFile.txt");
```

```
System.out.println(existingDirFile.isFile());
```

```
FileReader fr = new FileReader(existingDirFile);
```

```
BufferedReader br = new BufferedReader(fr);
```

```
String s;
```

```
while( (s = br.readLine()) != null)  
System.out.println(s);
```

```
br.close();
```

The File class.....key methods

<code>boolean canRead()</code>	Tests whether a file is readable or not
<code>boolean canWrite()</code>	Tests whether a file is modifyable or not
<code>boolean createNewFile()</code>	Creates a new empty file if it does not exist
<code>boolean exists()</code>	Tests whether a file or directory exists
<code>String getAbsolutePath()</code>	Returns the absolute pathname of a filesystem element
<code>String getName()</code>	Returns name of the file or directory
<code>boolean isDirectory()</code>	Tests whether a filesystem element is a directory
<code>boolean isFile()</code>	Tests whether a filesystem element is a file
<code>long length()</code>	Returns the size of a file in bytes
<code>String[] list()</code>	Returns the contents of a directory as an array
<code>boolean mkdir()</code>	Creates a directory

New methods added in File class

Method name	Description
<code>public boolean setWritable(boolean writable,boolean ownerOnly)</code>	A convenience method to set the owner's write permission for this abstract pathname.
<code>public boolean setReadable(boolean readable,boolean ownerOnly)</code>	Sets the owner's or everybody's read permission for this abstract pathname.
<code>public boolean setExecutable(boolean executable,boolean ownerOnly)</code>	A convenience method to set the owner's execute permission for this abstract pathname.

New methods added in File class

Method name	Description
<code>public boolean canExecute()</code>	Tests whether the application can execute the file denoted by this abstract pathname.
<code>public long getTotalSpace()</code>	Returns the size of the partition named by this abstract pathname.
<code>public long getFreeSpace()</code>	Returns the number of unallocated bytes in the partition named by this abstract path name.
<code>public long getUsableSpace()</code>	Returns the number of bytes available to this virtual machine on the partition named by this abstract pathname.

Buffered streams for optimization

- Wrap other streams
- Apply the Decorator pattern
- Primarily used for efficient high volume read and write operations
- Allow stream markers

Buffered streams.....quick look

```
/* wrap a file input stream inside a buffered input stream */  
BufferedInputStream bufferedInputStream;
```

```
bufferedInputStream = new BufferedInputStream(  
new FileInputStream("lib.dll"));
```

```
/* wrap a file output stream inside a buffered output stream  
*/
```

```
BufferedOutputStream bufferedOutputStream;
```

```
bufferedOutputStream = new BufferedOutputStream(  
new FileOutputStream("lib.so"));
```

Buffered streams.....quick look (Contd.)

```
/* wrap a file reader inside a buffered reader with a buffer  
size of 256 characters*/
```

```
BufferedReader bufferedReader;
```

```
bufferedReader = new BufferedReader(  
new FileReader("fiction.txt"), 256);
```

```
BufferedWriter bufferedWriter;
```

```
bufferedWriter = new BufferedWriter(  
new FileWriter("mystery.txt"), 256);
```


Buffered streams.....special

```
/* wrap a file reader inside a buffered reader with a buffer  
size of 256 characters */
```

```
BufferedReader bufferedReader;  
bufferedReader = new BufferedReader(  
new FileReader("fiction.txt"), 256);
```

```
// read something
```

```
// read some more
```

```
/* mark your current point in the stream and set the read  
ahead limit */
```

```
bufferedReader.mark(100);
```

```
// read something
```

```
// read some more but no more than 100 characters
```

```
/* reset the internal stream pointer */
```

```
bufferedReader.reset();
```

Bridging the Gap

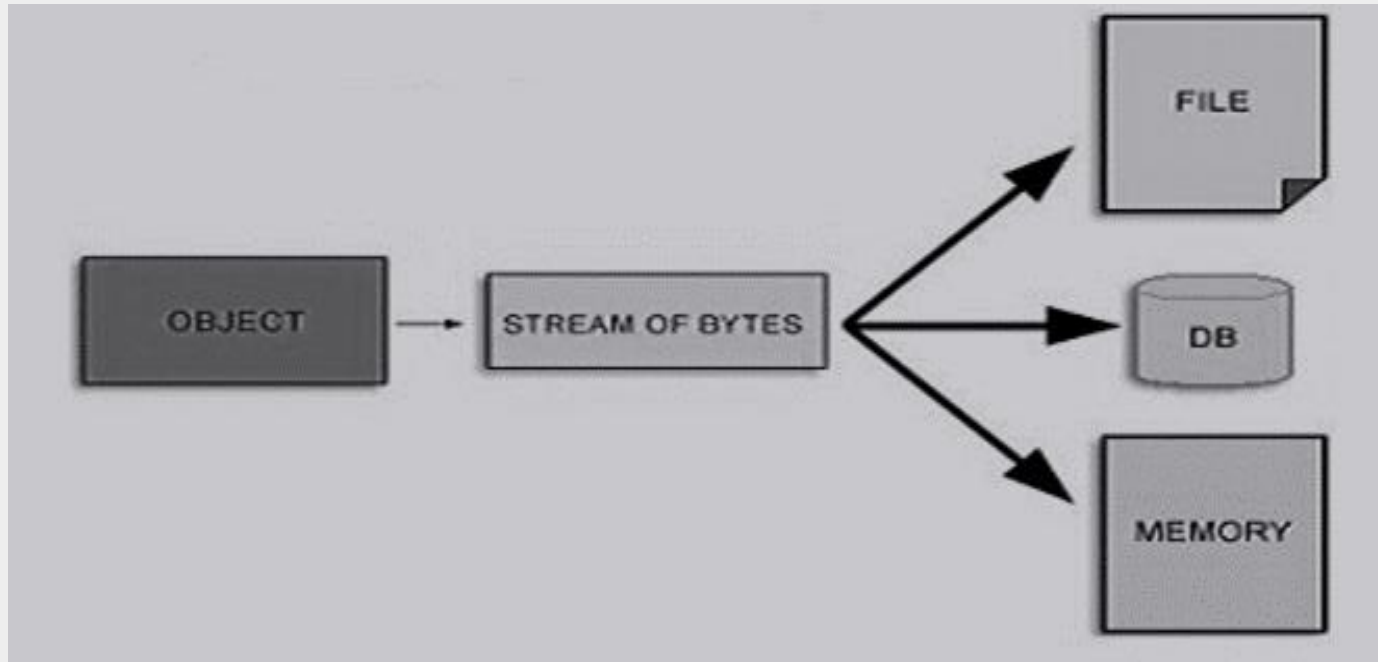
- Sometimes you need to bridge across the two hierarchies
 - Use `InputStreamReader` or `OutputStreamWriter`
- `InputStreamReader`
 - Reads bytes from an `InputStream`, and turns them into characters using a character encoding
- `OutputStreamWriter`
 - Turns characters sent to the `Writer` into bytes written by the `OutputStream`, again using a character encoding.

Serialization

- What is serialization ?
- Serialization is the process of saving the state of an object to a byte stream.
- It preserves the state of all fields except those marked transient or static and non-serializable fields inherited from the superclass
- It is implemented through the `java.io.Serializable` interface.

Serialization

- Use `java.io.ObjectOutputStream` to serialize
- `Serializable` is a marker interface
- Custom serialization implemented through `java.io.Externalizable` interface



How to serialize an object to a file?

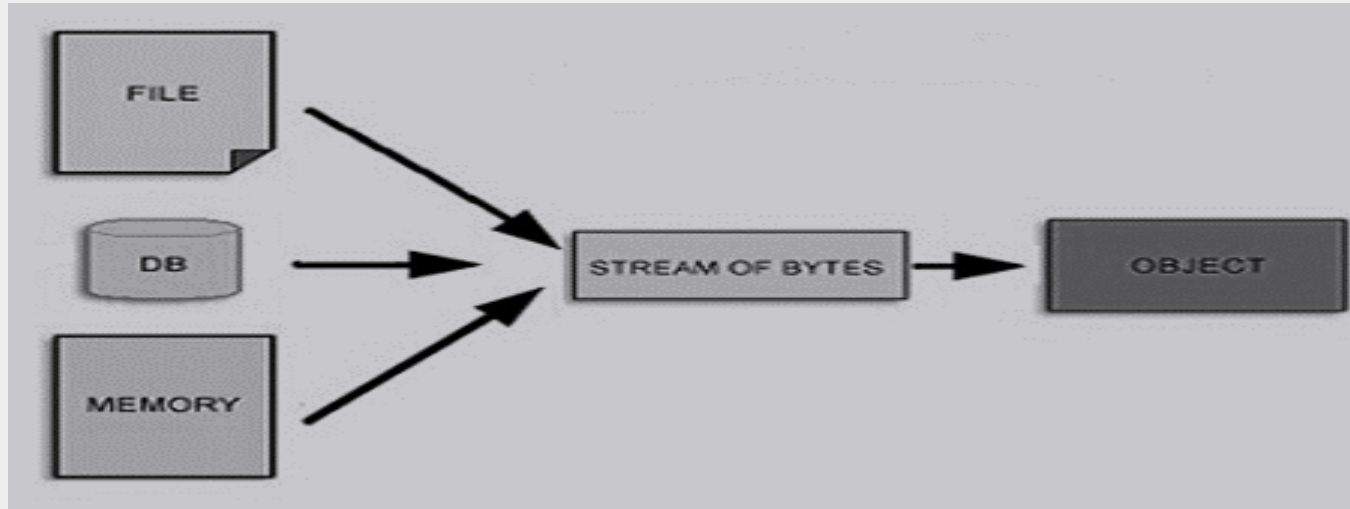
```
class Engine implements Serializable {  
  
    private static final long serialVersionUID =  
        0x000100001L;  
  
    private int type;  
    private double capacity;  
    public Engine() {  
  
    }  
  
    public Engine(int type, double capacity) {  
        this.type = type;  
        this.capacity = capacity;  
  
    }  
  
    // getter and setter methods  
}
```

How to serialize object to file?

```
public class TestSerialization {  
  
    public static void main(String[] args) {  
  
        Engine e1=new Engine(1,1000);  
        serializeEngine("engine.ser", e1);  
  
    }  
  
    public static void serializeEngine(String fileName,  
    Engine... engines) throws FileNotFoundException,  
    IOException{  
  
        ObjectOutputStream out=new ObjectOutputStream(new  
        FileOutputStream(fileName));  
  
        for (Engine engine : engines) {  
            out.writeObject(engine);  
        }  
    }  
}
```

De-serialization

- What is de-serialization ?



- De-serialization is the process of restoring the state of an object from a byte stream.
- Use `java.io.ObjectInputStream` to de-serialize

How to de-serialize object from file?

```
public class TestSerialization {  
  
    public static Engine deserializeEngine(String fileName)  
    throws ClassNotFoundException,  
    FileNotFoundException, IOException  
    {  
        ObjectInputStream in = new ObjectInputStream(new  
            FileInputStream(fileName));  
  
        Engine engine=(Engine) in.readObject();  
  
        return engine;  
    }  
}
```


Serialization with Composition Example

```
public class Engine implements Serializable {  
  
    private static final long  
        serialVersionUID = 0x000100001L;  
  
    private int type;  
  
    private double capacity;  
  
    public Engine(int type, double capacity) {  
        this.type = type;  
        this.capacity = capacity;  
    }  
  
    public int getType()        { return type; }  
  
    public double getCapacity(){ return capacity;}  
}
```

Serialization with composition sample....

here... Engine class
must have Implemented
Serializable Interface.
Otherwise It will throw
NotSerializableExceptio
n

```
public class Car implements Serializable {  
  
    private static final long serialVersionUID =  
        0x000200001L;  
  
    private long registrationNumber;  
  
    private Engine engine;  
  
    public Car(long registrationNumber, Engine engine)  
    {  
  
        this.registrationNumber = registrationNumber;  
        this.engine = engine;  
  
    }  
    public long getRegistrationNumber()  
    {  
        return registrationNumber;  
    }  
  
    public Engine getEngine() { return engine; }  
}
```

Serialization with composition sample....

```
Engine engine = new Engine(11,33.4);  
Car car = new Car(9876788L, engine);
```

```
try {  
    FileOutputStream fs;  
    ObjectOutputStream os;  
  
    fs = new FileOutputStream("testSer");  
    os = new ObjectOutputStream(fs);
```

```
    os.writeObject(car);  
    os.close();
```

```
    FileInputStream fis;  
    ObjectInputStream ois;
```

```
    fis = new FileInputStream("testSer");  
    ois = new ObjectInputStream(fis);
```

```
    car = (Car) ois.readObject();  
    ois.close();
```

```
} catch (Exception e) { e.printStackTrace(); }
```

Excluding field with transient Keyword

Excluding field
from serialization
with the help of
transient
keyword

```
public class Car implements Serializable {  
  
    private long registrationNumber;  
  
    private Engine engine;  
  
    private transient String color;  
  
    public Car(long registrationNumber, Engine engine) {  
  
        this.registrationNumber = registrationNumber;  
        this.engine = engine;  
    }  
  
    public long getRegistrationNumber() {  
        return registrationNumber;  
    }  
  
    public Engine getEngine() { return engine; }  
  
}
```

Serialization.....the other side

- What is written to the stream ?
 - Class of the object
 - Signature of the class
 - Values of all non-transient and non-static fields
 - Other referenced objects
 - A serial version unique identifier

Inheritance and Serialization

- If super-class is serializable then all sub-classes are by default serializable.
- If superclass is not serializable but sub-class is serializable
 - The super-class must have a no-argument constructor accessible to the subclass.
 - This constructor will be used to initialize the sub-class' sub-object.

Key points in serialization

- Serialization applies only to Objects.
- Static variables are class variables hence will not be serialized.
- Default serialization ties you to a class's current implementation.
- Forgetting the *serialVersionUID* can be painful while de-serializing.

Automatic Resource Management(ARM)

- While using resources like files, database, socket etc, they need to be closed explicitly.
- This needs to be specified in finally block.
- Java SE 7 answered to the automatic resource management problem in the form of a new language construct *try-with-resources statement*.
- Try with resource allows us to implicitly close the resource.

How is this achieved?

- An interface `AutoCloseable` is added to `java.lang` package in JDK7.
- Any resource that implements `AutoCloseable` interface can be used in try with resource statement.
- The `close()` method in `AutoCloseable` interface gets called implicitly when try block execution is completed.

How to use Try with resource?

```
import java.io.InputStream;
public class Test {

    private static void printFile() throws IOException {

        InputStream stream = null;
        try{
            stream = new FileInputStream("read.txt");
            int data = stream.read();
            while((data= stream.read()) != -1){
                System.out.println((char)data);
            }
        } finally{
            if(stream != null){
                stream.close();
            }
        }
    }

    public static void main(String[] args) throws IOException
    {
        new Test().printFile();
    }
}
```

Try-with-resources Statement

- Even-if any exception gets thrown in try block, the propagated exception will be the one thrown in finally block(if that operation fails).
- The resource variable will be declared inside the parentheses after try keyword.
- When try block execution is completed, the resource will be closed implicitly.
- The exception thrown in try will be propagated.

Try-with-resources Statement Syntax

- The syntax for specifying try with resource is as follows

```
try(FileInputStream input = new FileInputStream("file.txt") )  
{  
    //Code to be executed  
  
}  
catch(Exception ex) {  
    // code to be excuted  
  
}
```

Multiple resources

- Multiple resources can also be declared in try statement.

```
try(  
    FileInputStream input = new  
        FileInputStream("file.txt");  
  
    BufferedInputStream bufferedInput =  
        new BufferedInputStream(input)  
)  
{  
  
    //code to be executed  
}  
catch(Exception ex){  
  
    //code to be executed  
}
```

Console class

- Added in java.io package in JDK 1.6
- Provides the methods to access the character-based console device, associated with the current JVM.
- Refers the console associated with operating system.

Console class

- Provides facility to read text & password.
- Reads the password from the console without displaying it to the user.
- `readPassword()` method provides this facility.

How to get object of Console class?

- A static method , console() of System class returns object of Console class

```
Console console = System.console();
```


How to use Console class

```
import java.io.Console;

public class ConsoleDemo {

    public static void main(String[] args) {

        Console console = System.console();
        String name = console.readLine("Enter name:
");

        System.out.println(name);

        char ch[] = console.readPassword("Enter
password: ");

        System.out.println(ch);

    }
}
```

Summary

With this we have come to an end of our session, where we discussed about

- IO steams
- How to use IO Streams
- Serialization, Deserialization
- File class
- Console class

Appendix

A decorative graphic consisting of a horizontal orange line that extends from the left edge of the page. This line meets a vertical orange line that extends downwards to the bottom edge. At the point where they meet, a large orange circle is drawn, with its center at the intersection. The circle's top edge is near the top of the page, and its right edge is near the right edge of the page.

References

Key Contacts

Reference Material : Websites & Blogs

- http://www.tutorialspoint.com/java/java_files_io.htm
- <http://www.javatpoint.com/java-io>
- <https://docs.oracle.com/javase/tutorial/essential/io/streams.html>

Reference Material : Books

- **Head First Java**
 - By: Kathy Sierra, Bert Bates
 - Publisher: O'Reilly Media, Inc.
- **Java Complete Reference**
 - By Herbert Schildt



Thank you!

Persistent Interactive | Persistent University

